Performance and Portability in Multi-GPU Branch-and-Bound: Chapel *Versus* CUDA and HIP for Tree-Based Optimization

Ivan Tagliaferro[*†] Guillaume Helbecque[*]
Ezhilmathi Krishnasamy[†] Nouredine Melab[*] Grégoire Danoy[†]

1 Introduction

Tree search algorithms, such as the Branch-and-Bound (B&B) method, are essential tools in exact combinatorial optimization. Parallel B&B presents significant challenges in achieving scalability arising due to irregular and fine-grained workload associated and the hardware heterogeneity of modern supercomputers [1]. This work focuses on leveraging GPU heterogeneity in the HPC environment within exact tree search optimization algorithms, through a comparison between a proposed CUDA-based low-level implementation and a high-level PGAS-based one. We revisit the design and implementation of a pool-based GPU-accelerated parallel B&B algorithm tailored for heterogeneous CPU / GPU systems originally developed in Chapel. A low-level counterpart implementation, suited for Nvidia and AMD GPU architectures, is proposed for performance gain, appeasing some of the portability issues usually related to low-level implementations. This comparison provides insights into the trade-offs between different programming paradigms in large-scale, heterogeneous computing environments.

2 Background

The B&B method is a tree-based search method suited to solving NP-hard optimization problems. The tree nodes represent partial or complete solutions to a problem. The method revolves around four key operators: **branching**, **selection**, **bounding**, and **pruning**. We combine two **selection** strategies, *Breadth-First Search* (BFS), a queue container (FIFO), and

[*]Université de Lille, CNRS/CRIStAL UMR 9189, Centre Inria de l'Université de Lille, France.
[†]University of Luxembourg, DCS-FSTM/SnT, Belval, Luxembourg.

Depth-First Search (DFS), a stack container (LIFO). Two of the main parallelization models for tree search algorithms are employed: *Parallel Tree Exploration* (PTE) and *Parallel Evaluation of Bounds* (PEB) [2]. Multiple works tackled large-scale optimization problems, most notably [3–5]. These are limited by problem-specific design and often make use of high-complexity implementation. In contrast, PGAS-based approaches implemented in Chapel language proposed high-level generic implementations, leveraging CPU or CPU+GPU computations. Most notably, [6, 7] dealt with GPU-accelerated B&B, applying it to the N-Queens problem. Our work complements [8] by proposing a counterpart CUDA-based version implemented in C of its pool-based GPU-accelerated parallel B&B applied to the Permutation Flowshop Scheduling Problem (PFSP).

3 Implementation

An initial BFS executed on a single CPU thread generates nodes stacked in an initial pool. The work pool is a data structure that holds unprocessed nodes, facilitating insertion and retrieval. Each node contains the necessary information for its evaluation. Two parameters, m and M, are used to fine-tune work offloading onto the GPUs. When the number of nodes is greater than $m \times nb\_CPU\_Threads$, nodes are cyclically distributed among multiple pools, each managed by a CPU thread. In this accelerated bounding phase (PEB), nodes from the multi-pools are sent for evaluation by the GPUs, up to M nodes per GPU kernel call, while CPU threads continue branching (PTE) and pruning using a DFS strategy to mitigate memory contention. GPU tasks are managed *via* CUDA API calls on the single-GPU version. The multi-GPU version integrates an OpenMP layer that controls multiple CPU threads at the intra-node level, allowing multi-GPU use. During the multi-GPU exploration, a dynamic load balancing is achieved through a work-stealing (WS) mechanism between pools. Each pool is equipped with an atomic lock (C11 standard) for proper workload memory management during WS. Our WS is a *strategy on request* based on **random victim selection**, **locked pools** (spin-lock), and a **steal-half strategy**. Finally, a termination detection determines when the GPU work is complete, allowing the algorithm to evaluate any remaining nodes using a CPU thread (below the threshold of m). Every worker counts with an atomic state variable. The states are updated based on activity and WS occurrences. The termination condition is satisfied when all CPU threads are idle, *i.e.*, no further work is treated, or successful WS attempts occur.

4 Experimental Evaluation

We use the Grid'5000 large-scale testbed to evaluate the single-GPU (SG-B&B) and single-node multi-GPU (MG-B&B) implementations on both Nvidia and AMD GPU architectures. We apply them to the PFSP, a well-known problem in the operational research community, consisting of scheduling a set of n jobs across a series of m machines arranged in a fixed sequence. The algorithm's upper bound is initialized with the optimal known solution, ensuring the exploration of the same critical tree. The benchmark instances proposed by Taillard [9] with 20 jobs and 20 machines are used, $ta021$ to $ta030$. The lower bound function used is the one commonly referred to as LB2 proposed by [10]. The HIP version is obtained using the *hipify-perl* tool. The software versions are Chapel 2.1.0, C compiler 10.2.1, CUDA 11.2, OpenMP 4.5, and ROCm/HIP 4.5.0. The parameters are set to $m = 25$ and $M = 50000$. The experiments were carried out on the following GPU architectures:

- *Nvidia V100*: 20-core Intel Xeon E5-2698 v4 (Broadwell) @ 2.20 GHz and 512 GiB of memory, equipped with 8 Nvidia Tesla V100-SXM2-32GB GPUs (32 GiB HBM2 memory and 5120 CUDA cores @ 1230 MHz);
- *AMD MI50*: 48-core AMD EPYC 7642 (Zen 2) @ 2.3 GHz and 512 GiB memory, equipped with 8 AMD Radeon Instinct MI50 GPUs (32 GB HBM2 memory and 3840 stream processors @ 1200 MHz).

SG-B&B Results: CUDA-based implementation performs consistently better than Chapel, being up to 10% faster. Chapel's performance is better than that of CUDA in the smallest workloads in Nvidia architecture due to internal optimizations in its GPU kernel calls. On the other hand, HIP outperforms Chapel in all tested instances, being 48% to 54% faster. This disparity in both comparisons is due to at least two factors: (1) The Chapel's support for AMD GPUs is recent and potentially less optimized; (2) Chapel compiler may fail to take advantage of register coalescing optimizations provided by LLVM, resulting in higher register pressure and, thus, lower warp occupancy, reducing parallel execution [11].

MG-B&B Results: The performance of the MG-B&B implementation is evaluated through speed-up studies comparing CUDA/HIP and Chapel implementations across 2, 4 and 8 GPUs. For clarity and ease of interpretation, we selected four representative instances with similar speed-up behavior: $ta030$, $ta023$, $ta025$, and $ta021$. Fig. 1a. illustrates the evolution of speed-ups when comparing CUDA and Chapel implementations on the Nvidia architecture. Across all instances, Chapel speed-ups scale linearly and consistently, while CUDA's speed-ups have scalable growth when increasing instance size. For problems less computationally demanding, penalties asso-

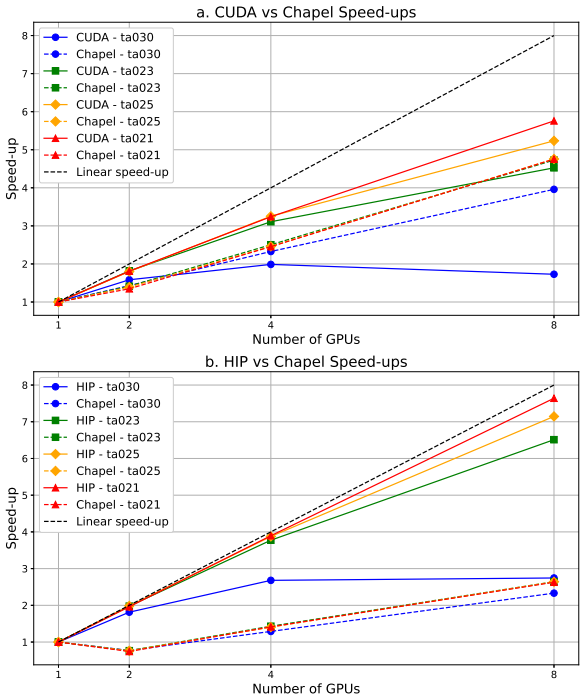

Figure 1: Speed-up comparison between CUDA/HIP and Chapel implementations.

ciated with memory and synchronization APIs in CUDA become more pronounced. However, for increased computational workload, CUDA's SMIT approach performs well. Fig. 1b. exhibits HIP's near-optimal scalability as the number of GPUs increases for larger problem instances. HIP APIs, while functional, are not exclusive for AMD GPU architecture as CUDA's are for Nvidia, leading to less efficient workload deployment onto GPUs. Consequently, HIP benefits more from increasing the number of GPUs. However, the strong scalability exhibited by HIP may obscure underlying issues related to the optimization of kernel calls, since it is a single source code that applies to both architectures.

Chapel's PGAS paradigm significantly improves productivity, particularly in the context of memory management and GPU use. However, this higher level of abstraction comes with certain limitations in manual tuning capabilities, such as fine-grained optimization in GPU kernel calls (*e.g.* thread indexing). This trade-off between fine control and productivity reflects Chapel's emphasis on ease of use and portability, while still offering efficient execution between different platforms.

5 Conclusions and Future Works

In this work, we revisit the design and implementation of a pool-based GPU-accelerated parallel B&B algorithm suited for Nvidia and AMD GPUs. Single- and multi-GPU versions were implemented in CUDA/HIP and compared to their counterparts based on Chapel. Although Chapel offers higher abstraction, portability across Nvidia and AMD GPUs, and provides competitive results in relation to its CUDA counterpart for smaller instances, its slower execution times in larger instances highlight its need for further optimizations. In the future, we aim to extend our B&B implementation to operate at the inter-node level. Metaheuristics can also be applied for fine-tuning GPU and WS parameters, aiming for the optimality of unresolved combinatorial optimization problems.

Acknowledgment

This work was partially supported by the French government through the program "France 2030" (SFRI project GRAEL/ ANR-21-SFRI-005, PI: Université de Lille) managed by the National Research Agency; by the Agence Nationale de la Recherche (ref. ANR-22-CE46-0011) and the Luxembourg National Research Fund (FNR) (ref. INTER/ANR/22/17133848), under the UltraBO project; and by the FNR POLLUX program under the SERENITY project (ref. C22/IS/17395419). Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`). All code written in support of this publication is publicly available on GitHub at `https://github.com/Guillaume-Helbecque/GPU-accelerated-tree-search-Chapel`.

References

[1] Top500 Project, "Top500 International Ranking," https://www.top500.org/, 2024, accessed: 2024-08-12.

[2] B. Gendron and T. G. Crainic, "Parallel Branch-And-Bound Algorithms: Survey and Synthesis," *Operations Research*, vol. 42, no. 6, pp. 1042–1066, 1994.

[3] I. Chakroun, N. Melab, M. Mezmaz, and D. Tuyttens, "Combining multi-core and GPU computing for solving combinatorial optimization problems," *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1563–1577, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731513001615

[4] T.-T. Vu and B. Derbel, "Parallel Branch-and-Bound in multi-core multi-CPU multi-GPU heterogeneous environments," *Future Generation Computer Systems*, vol. 56, pp. 95–109, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X15003222

[5] J. Gmys, "Exactly Solving Hard Permutation Flowshop Scheduling Problems on Peta-Scale GPU-Accelerated Supercomputers," *INFORMS Journal on Computing*, vol. 34, pp. 2502–2522, 9 2022.

[6] T. Carneiro, E. Kayraklioglu, G. Helbecque, and N. Melab, "Investigating Portability in Chapel for Tree-Based Optimization on GPU-Powered Clusters," in *Euro-Par 2024: Parallel Processing: 30th European Conference on Parallel and Distributed Processing, Madrid, Spain, August 26–30, 2024, Proceedings, Part III*. Berlin, Heidelberg: Springer-Verlag, 2024, p. 386–399. [Online]. Available: https://doi.org/10.1007/978-3-031-69583-4_27

[7] G. Helbecque, E. Krishnasamy, N. Melab, and P. Bouvry, "GPU-Accelerated Tree-Search in Chapel Versus CUDA and HIP," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops*, 05 2024, pp. 872–879.

[8] G. Helbecque, E. Krishnasamy, T. Carneiro, N. Melab, and P. Bouvry, "A Chapel-based Multi-GPU Branch-and-Bound Algorithm," in *Proceedings of Euro-Par Workshops*, Madrid, Spain, Aug 2024.

[9] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993, project Management anf Scheduling.

[10] B. J. Lageweg, J. K. Lenstra, and A. H. G. R. Kan, "A General Bounding Scheme for the Permutation Flow-Shop Problem," *Operations Research*, vol. 26, no. 1, pp. 53–67, 1978.

[11] J. Milthorpe, X. Wang, and A. Azizi, "Performance portability of the chapel language on heterogeneous architectures," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2024, pp. 6–13.